Advanced Java Programming: (17625)

Mr. Nilesh Vishwasrao Patil

Teaching and Examination Scheme

Teaching and Examination Scheme

Teaching Scheme			Examination Scheme					
TH	TU	PR	PAPER HRS	TH	PR	OR	TW	TOTAL
03	--	04	02	100#*	50#	--	50@	200

Mr. Nilesh Vishwasrao Patil

AJP: Chapter

Chapter #	Chapter Name	Marks (As per Syllabus)
01	**Introduction to Abstract Windowing Toolkit(AWT) & Swings**	24
02	**Event Handling**	20
03	**Networking & Security**	16
04	**Interacting with Database**	20
05	**Servlets & JSP**	20

Mr. Nilesh Vishwasrao Patil

Introduction to Abstract Windowing Toolkit (AWT) & Swings

24 Marks

Specific Objectives

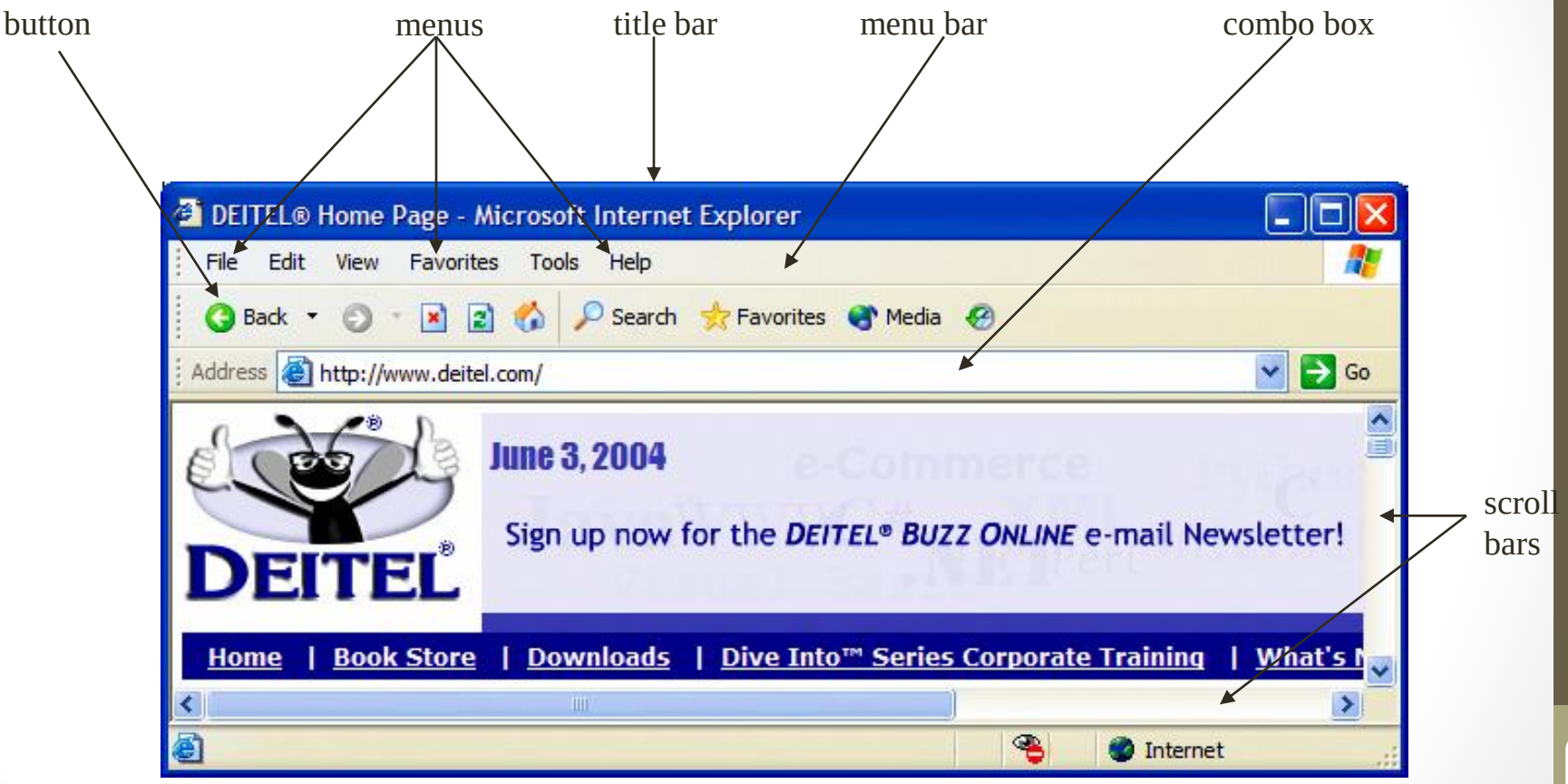
- To design and develop Graphical User Interface (GUI) programs using AWT and Swing component.
- To arrange the GUI components using different layout.

Mr. Nilesh Vishwasrao Patil

GUI (Graphical User Interface)

- GUI offers user interaction via some graphical components.
- Window, frame, Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc.
- Using these components we can create an interactive user interface for an application.
- GUI provides result to end user in response to raised event.
- GUI is entirely based event.

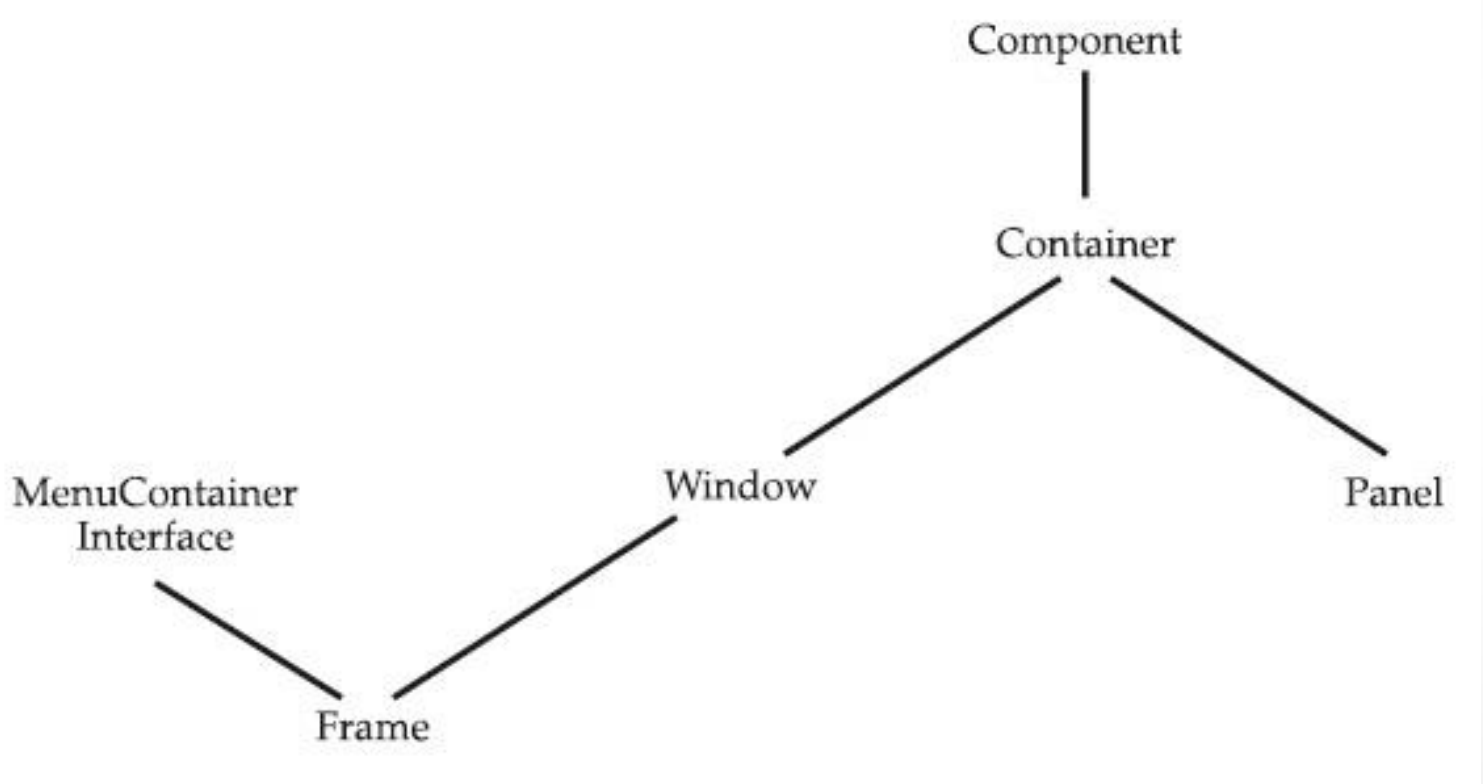
GUI (Graphical User Interface)



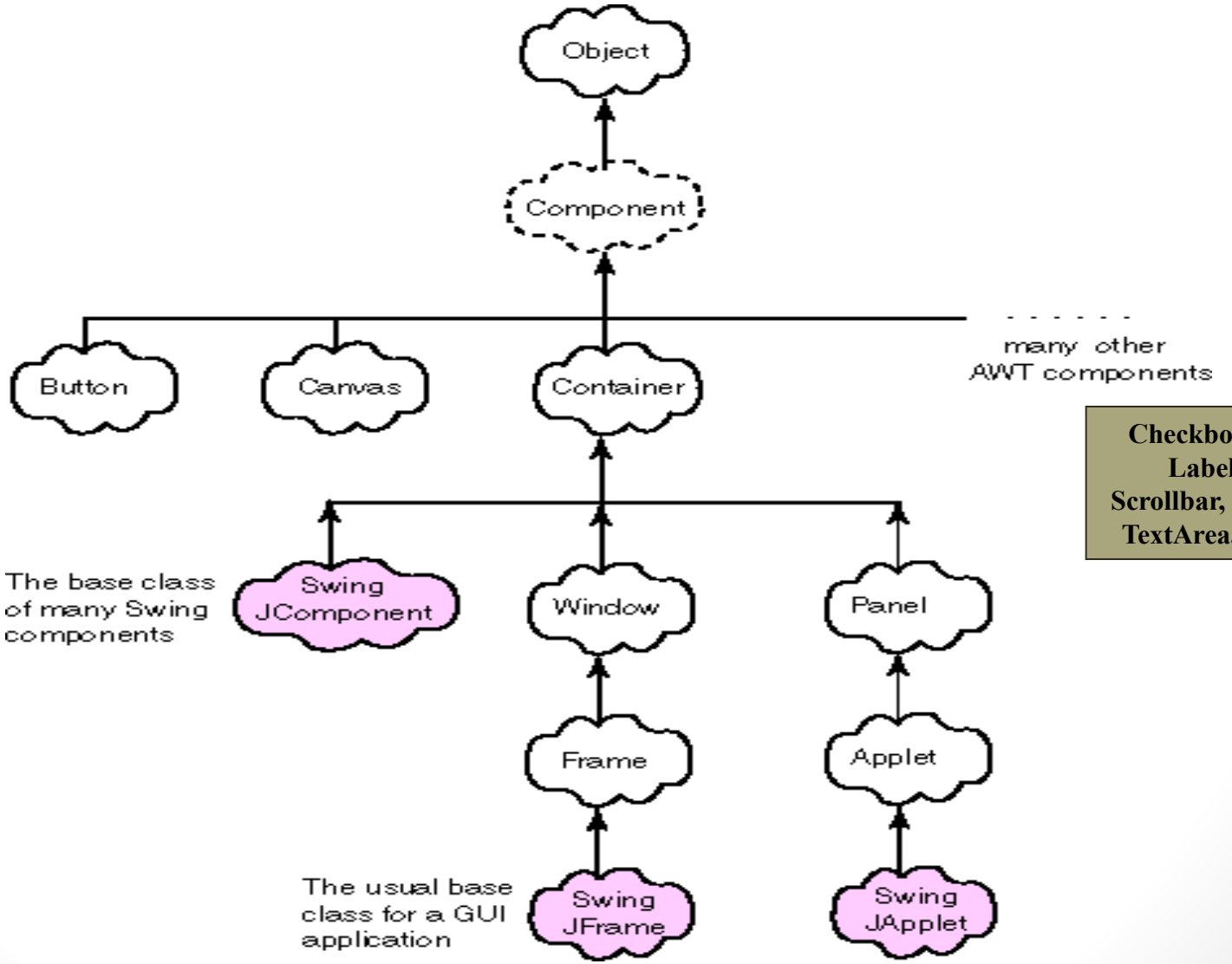
AWT (Abstract Window Toolkit)

- AWT contains numerous classes and methods that allow you to create and manage window.
- import java.awt.*;
- **Java AWT** is *an API to develop GUI or window-based application in java*.
- *Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.*
- AWT is heavyweight i.e. its components uses the resources of system.

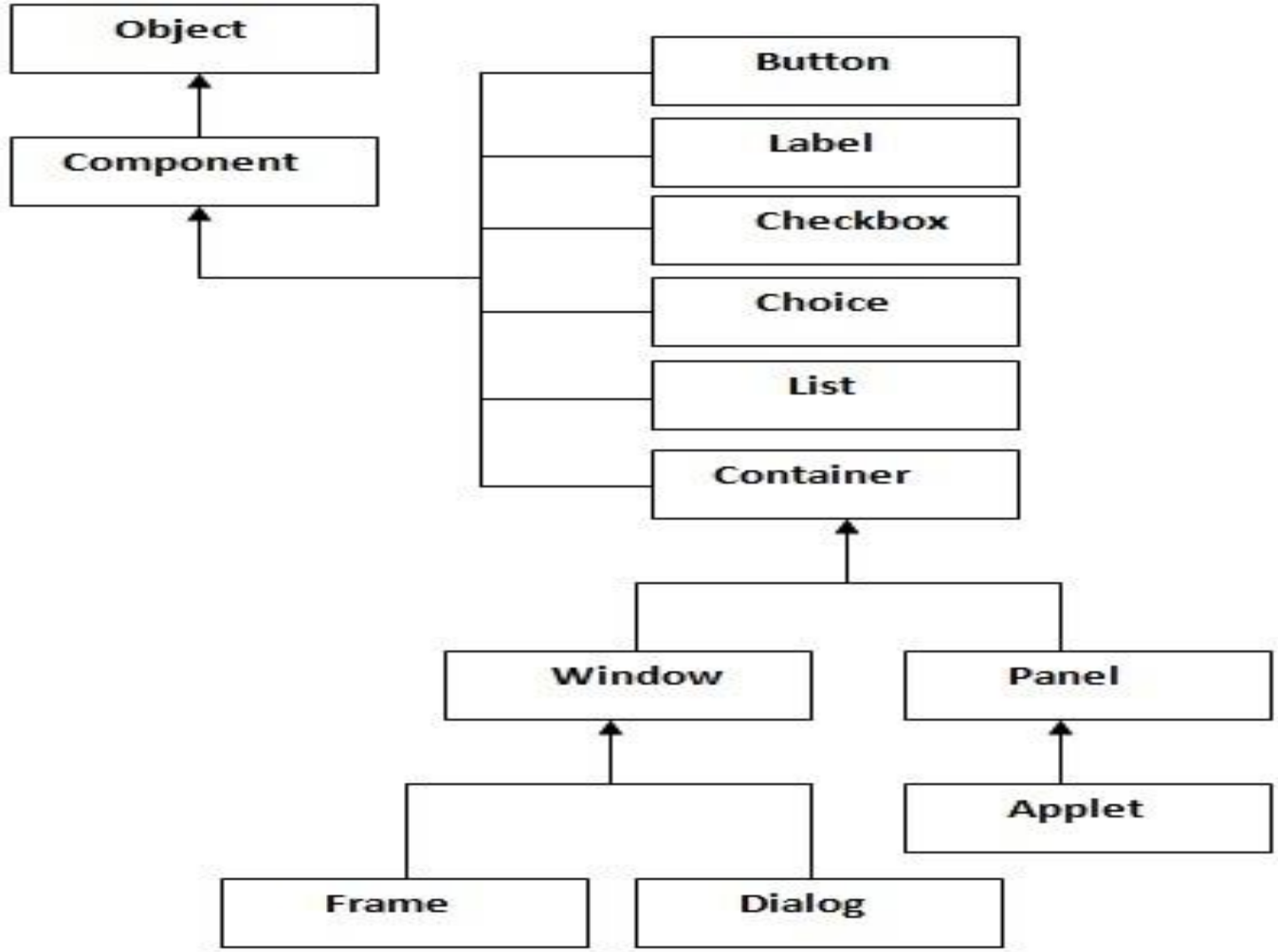
AWT Class Hierarchy



AWT Class Hierarchy : Detailed



AWT Class Hierarchy : Detailed



Component

- Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.
- At the top of the AWT hierarchy is the **Component** class.
- It is abstract class that encapsulates all of the attributes of a visual component.
- It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting.

Component

- **Which** object is responsible for remembering the current foreground and background colors and the currently selected text font? (Answer: Componenet)
- It extends: Object Class
- Implements: ImageObserver, MenuContainer, Serializable

Container

- **Container** class is a subclass of **Component**.
- Provided additional methods that allow other **Component** to place on it.
- **Which (Container)** is responsible for laying out (that is, positioning) any components that it contains.
- It does this through the use of various layout managers.

Container

- Container is a component in AWT that can contain another components like buttons, text fields, labels etc.
- The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Containers and Components

- The job of a Container is to hold and display Components
- Some common subclasses of Component are Button, Checkbox, Label, Scrollbar, TextField, and TextArea
- A Container is also a Component
- Some Container subclasses are Panel (and Applet, JApplet), Window (Frame, JFrame)

Panel

- **Panel** is a concrete subclass of **Container.**
- It doesn’t add any new methods which simply implements **Container**.
- **Panel** is the immediate superclass of **Applet**. When screen output is directed to an applet, it is drawn on the surface of a **Panel** object.

Panel

- **Panel** is one type of container that does not contain a title bar, menu bar, or border.
- When you run an applet using an applet viewer, the applet viewer provides the title and border.
- Uses add() to add component defined by container.
- Uses setSize(), setLocation(), setBounds() defined by Component.

An Applet is Panel is a Container

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----java.awt.Panel
                            |
                            +----java.applet.Applet
```

…so you can display things in an Applet

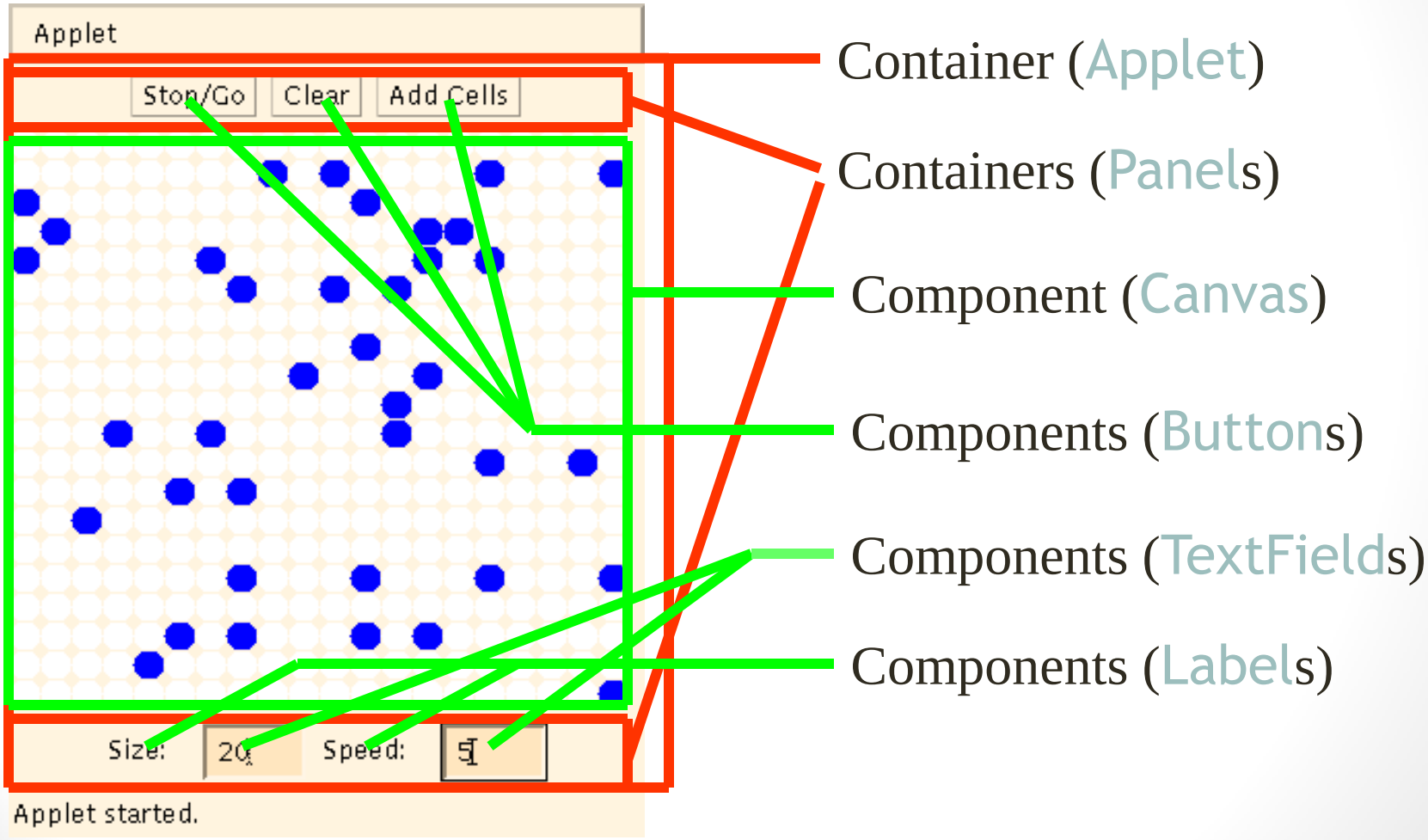
Applets

- Applet is a public class which is predefined by java.applet.Applet.
- There is no main() method in Applet like Application program. The main() method is defined by browser or Appletviewer for Applet.
- Life cycle methods: init, start, paint, stop, destroy
- Applet is one type of container and subclass of Panel. Applet is superclass of JApplet

To create an applet

- import java.applet.*;
- Import java.awt.*;
- Applet code in comment.
- Extends Applet class
- Life cycle method
- Class must be public

Example: A "Life" applet



Window

- Creates top-level window means directly on the desktop.
- Don’t create Window objects directly.
- The window is the container that have no borders and menu bars.
- Uses Frame, Dialog class which is subclass of Window class for creating window.

Frame

- It is subclass of **Window** and has a title bar, menu bar, borders, and resizing corners.
- Frame object can be created from program or Applet.
- Through Applet: Warning message will display “Java Applet Window”.
- Through Program or Application: Normal window is created.

Working with Frame Window

- Extends Frame class
- Constructor are:
  - Frame()
  - Frame(String title)
- Setting and Getting window size:
  - void setSize(int width, int height)
  - void setSize(Dimension newsize)
  - Dimension getSize()
- Showing and Hiding Frame
  - void setVisible(boolean visibleFlag)

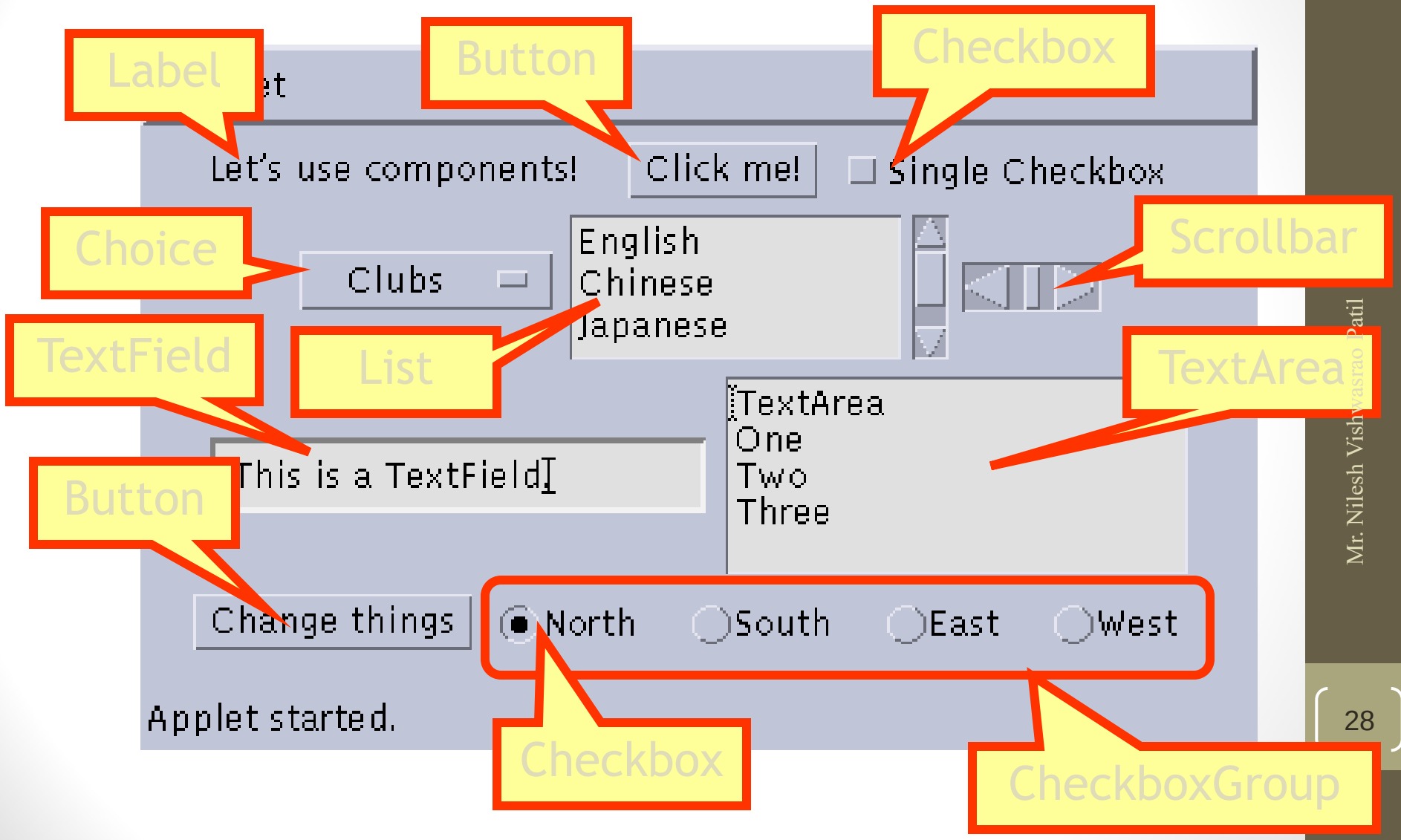
Working with Frame Window

- Setting title to frame
  - void setTitle(String title)
- Closing Frame: Four Ways
  - Implements WindowListener interface.
  - Extends WindowAdapter class.
  - WindowAdapter Inner class.
  - WindowAdapter anonymous inner classes.

AWT Controls and Layout Manager

- AWT Controls: Component which allows you to interact with application.
  - Labels
  - Button
  - Checkbox
  - Checkbox group
  - Scrollbars
  - Text field
  - Text Area

Some types of components



AWT Controls and Layout Manager

- Layout Manager: Positioning the components in the container.
  - Flow Layout
  - Border Layout
  - Grid Layout
  - Card Layout
  - Grid Bag Layout

- Menubars, menus, dialog boxes, file dialog.

AWT Controls:

- Allows user to interact with application.
- Adding the control in Window
  - First create instance of control.
  - Call add() method defined by container
  - Component add(Component compObj)
- Removing the Control
  - Call remove() method defined by container
  - void remove(Component obj)
  - For remove all: removeAll() method call.

AWT Control: Label

- Used to just display string on window.
- Passive Components
- Constructors:
  - Label( )
  - Label(String *str*) //left - justified
  - Label(String *str*, int *how*) // Label.LEFT, Label.RIGHT, Label.CENTER
- Methods to perform operation: Setter and Getter Method.
  - About text:
    - void setText(String *str*)
    - String getText( )
  - About Alighment
    - void setAlignment(int *how*)
    - int getAlignment( )

AWT Control: Button

- It contains a label and that generates an event when it is pressed.
- Active Components
- Constructors:
  - Button( )
  - Button(String *str*)
- Methods to perform operation: Setter and Getter Method.
  - void setLabel(String *str*)
  - String getLabel( )

AWT Control: Button Handling

- When Button pressed which generates an event when it is pressed.
- Implements ActionListener interface.
- Interface has defined actionPerformed() method, called when event is generated.
- ActionEvent object is supplied as argument to the method.
- ActionEvent object refers both Button and Label of Button
- Label will get by using **getActionCommand()** from ActionEvent which passed.

AWT Control: CheckBox

- Used to turn an option on or off.
- Small box: check mark or not.
- Each check box has label.
- Constructors are:
  - Checkbox( )
  - Checkbox(String *str*)
  - Checkbox(String *str*, boolean *on*)
  - Checkbox(String *str*, boolean *on*, CheckboxGroup *cbGroup*)
  - Checkbox(String *str*, CheckboxGroup *cbGroup*, boolean *on*)

AWT Control: CheckBox

- Setter and Getter methods:
  - boolean getState( )
  - void setState(boolean *on*)
  - String getLabel( )
  - void setLabel(String *str*)
- The value of on determine initial state (true/false).

AWT Control: CheckBox Handling

- Check box is selected or deselected, an item event is generated.
- For handling implements ItemListener interface
- ItemListener interface is defines **itemStateChanged( )** method.
- **ItemEvent** object is supplied as the argument.
- getState() : Get Status about checkbox.
- Following methods determine and set status:
  - Checkbox getSelectedCheckbox( )
  - void setSelectedCheckbox(Checkbox *which*)

AWT Control: Choice Class

- Used to create a pop-up list items.
- Default constructor Choice() create empty list.
- For add item in list and select active item:
  - void add(String name)
  - void select(int *index*)
  - void select(String *name*)
- Each item in the list is a string that appears as a left-justified label in the order it is added to the **Choice** object.
- To determine selected item:
  - String getSelectedItem( )
  - int getSelectedIndex( )
  - String getItem(int *index*)

AWT Control: Handling Choice

- When Choice selected, an item event is generated.
- Implements the **ItemListener** interface.
- Interface defines the **itemStateChanged( )** method.
- **ItemEvent** object is supplied as the argument to this method.

AWT Control: List

- **List** class provides a compact, multiple-choice, scrolling selection list.
- **List** object can be constructed to show any number of choices in the visible window.
- In Choice only one item is shown.
- Constructors
  - List( )
  - List(int *numRows*)
  - List(int *numRows*, boolean *multipleSelect*)

AWT Control: List

- Following methods are used to add items:
  - void add(String *name*)
  - void add(String *name*, int *index*)
- For single selection items:
  - String getSelectedItem( )
  - int getSelectedIndex( )
- For Multi selection items:
  - String[ ] getSelectedItems( )
  - int[ ] getSelectedIndexes( )

AWT Control: List

- To retrieve item:
  - String getItem(int *index*)
- To get Item Count
  - int getItemCount( )
- Active Item
  - void select(int *index*)

AWT Control: List Handling

- Two types of event generated:
  - For double clicked: ActionEvent generated.
  - For select and deselect item: ItemEvent generated.
- Implements ActionListener interface and ItemListener.

AWT Control: Scrollbar

- Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Used to select continuous values between a specified minimum and maximum.
- Scroll bars may be oriented horizontally or vertically.
- Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow.
- The current value of the scroll bar relative to its minimum and maximum values is indicated by the *slider box* for the scroll bar.
- The slider box can be dragged by the user to a new position.

AWT Control: Scrollbar

- Constructors:
  - Scrollbar( ) : construct new vertical scrollbar
  - Scrollbar(int *style*) : : construct new scrollbar with style orientation
  - Scrollbar(int *style,* int *initialValue*, int *thumbSize*, int *min*, int *max*)
- *style* : **Scrollbar.VERTICAL or Scrollbar.HORIZONTAL**
- For set Values:
  - void setValues(int *initialValue*, int *thumbSize*, int *min*, int *max*)
- For get and set current value:
  - int getValue( )
  - void setValue(int *newValue*)

AWT Control: Scrollbar

- For get Min and Max value:
  - int getMinimum( )
  - int getMaximum( )
- By default unit increment/decrement is 1 and Block page-up and page-down increment/decrement is 10.
- For change increment and decrement:
  - void setUnitIncrement(int *newIncr*)
  - void setBlockIncrement(int *newIncr*)

AWT Control: Handling Scrollbar

- **AdjustmentEvent** is generated.
- Implement the **AdjustmentListener** interface.
- adjustmentValueChanged() method we have to override
- **getAdjustmentType( )** method can be used to determine the type of the adjustment.
- BLOCK_DECREMENT: A page-down event has been generated.
- BLOCK_INCREMENT: A page-up event has been generated.
- TRACK: An absolute tracking event has been generated.
- UNIT_DECREMENT: user clicks in the right arrow of a horizontal scroll bar, or the bottom arrow of a vertical scroll bar
- UNIT_INCREMENT: User clicks in the left arrow of a horizontal scroll bar, or the top arrow of a vertical scroll bar.

AWT Control: TextField

- TextField is subclass of TextComponent. TextComponent is subclass of Component.
- **TextField** class implements a single-line text-entry area, usually called an *edit control.*
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- Constructors:
  - TextField( )
  - TextField(int *numChars*)
  - TextField(String *str*)
  - TextField(String *str*, int *numChars*)

AWT Control: TextField

- Setter and Getter Method of TextField and TextComponent:
  - String getText( )
  - void setText(String *str*)
- Particular Text selection:
  - String getSelectedText( )
  - void select(int *startIndex*, int *endIndex*)
- About Modification of Text:
  - boolean isEditable( )
  - void setEditable(boolean *canEdit*)

AWT Control: TextField

- Setting echo character to text field and related methods:
  - void setEchoChar(char *ch*)
  - boolean echoCharIsSet( )
  - char getEchoChar( )
- Button can be used to Handling Event: ActionEvent generates.
- Implements ActionListener Class

AWT Control: TextArea

- Need ? Sometimes a single line of text input is not enough for a given task.
- Subclass of TextComponent.
- Constructors:
  - TextArea( )
  - TextArea(int *numLines,* int *numChars*)
  - TextArea(String *str*)
  - TextArea(String *str*, int *numLines*, int *numChars*)
  - TextArea(String *str*, int *numLines*, int *numChars*, int *sBars*)

AWT Control: TextArea

- The values of sbar:
  - SCROLLBARS_BOTH
  - SCROLLBARS_NONE
  - SCROLLBARS_HORIZONTAL_ONLY
  - SCROLLBARS_VERTICAL_ONLY
- It supports: **getText( )**, **setText( )**, **getSelectedText( )**, **select( )**, **isEditable( )**, and **setEditable( )**
- Other some methods:
  - void append(String *str*)
  - void insert(String *str*, int *index*)
  - void replaceRange(String *str*, int *startIndex*, int *endIndex*)

AWT Control: TextArea

- The values of sbar:
  - SCROLLBARS_BOTH
  - SCROLLBARS_NONE
  - SCROLLBARS_HORIZONTAL_ONLY
  - SCROLLBARS_VERTICAL_ONLY
- It supports: **getText( )**, **setText( )**, **getSelectedText( )**, **select( )**, **isEditable( )**, and **setEditable( )**
- Other some methods:
  - void append(String *str*)
  - void insert(String *str*, int *index*)
  - void replaceRange(String *str*, int *startIndex*, int *endIndex*)

Arranging components : LM

- Layout means the arrangement of components within the container.
- Layout manager automatically positions all the components within the container.
- LayoutManager:
  - Defines the interface for classes that know how to lay out Containers.
- LayoutManager2:
  - It is the sub-interface of the LayoutManager. This interface is for those classes that know how to layout containers based on layout constraint object.

Arranging components : LM

- Every Container has a layout manager
- The default layout for a Panel and Applet is FlowLayout
- The default layout for a Window and Frame is a BorderLayout
- We could set it explicitly with: setLayout()

```
setLayout (new FlowLayout( ));
```

- You could change it to some other layout manager

Different Layout Manager

- FlowLayout
  - The FlowLayout is the default layout.It layouts the components in a directional/horizontally flow.
- BorderLayout
  - The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
- GridLayout
  - The GridLayout manages the components in form of a rectangular grid.
- CardLayout
  - The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
- GridBagLayout
  - This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size.

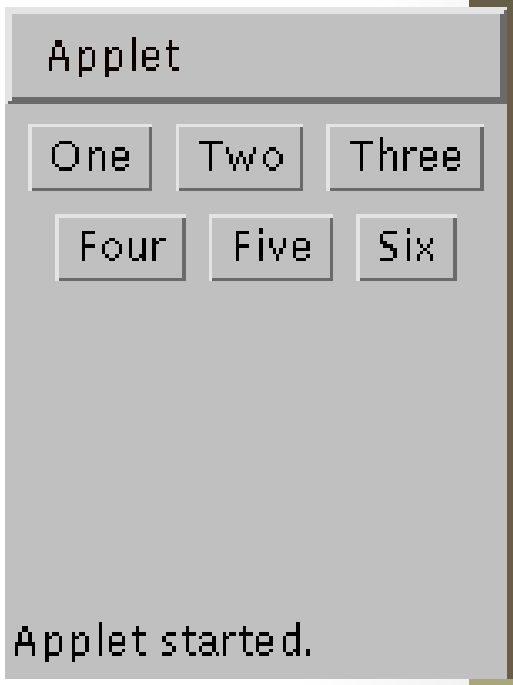
FlowLayout

- Use add(*component*); to add to a component when using a FlowLayout
- Components are added left-to-right
- If no room, a new row is started
- Exact layout depends on size of Applet
- Components are made as small as possible
- FlowLayout is convenient but often not good.

Complete example: FlowLayout

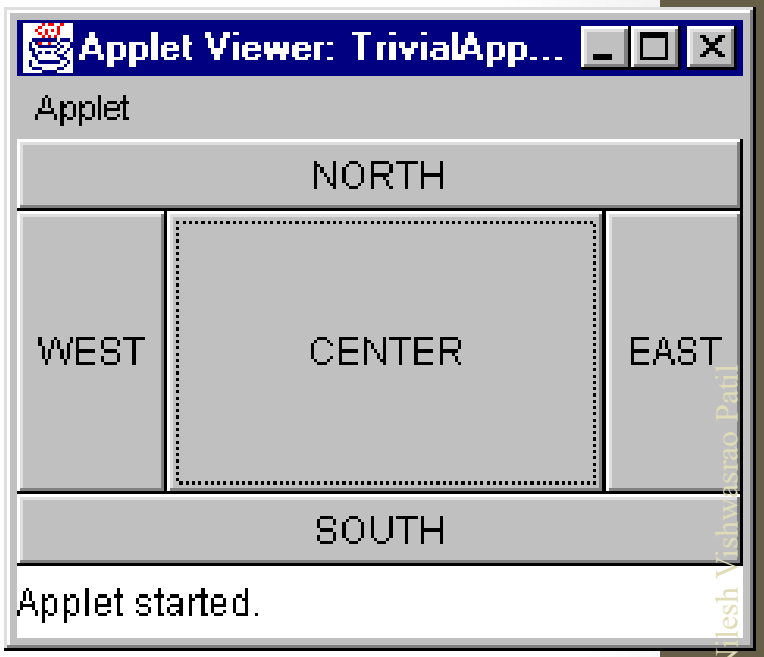
```
import java.awt.*;
import java.applet.*;

public class FlowLayoutExample extends Applet {
  public void init () {
    setLayout (new FlowLayout ()); // default
    add (new Button ("One"));
    add (new Button ("Two"));
    add (new Button ("Three"));
    add (new Button ("Four"));
    add (new Button ("Five"));
    add (new Button ("Six"));
  }
}
```



BorderLayout

- At most five components can be added
- If you want more components, add a Panel, then add components to it.
- setLayout (new BorderLayout());



add (new Button("NORTH"), BorderLayout.NORTH);

BorderLayout with five Buttons

```
public void init() {
   setLayout (new BorderLayout ());
   add (new Button ("NORTH"), BorderLayout.NORTH);
   add (new Button ("SOUTH"), BorderLayout.SOUTH);
   add (new Button ("EAST"), BorderLayout.EAST);
   add (new Button ("WEST"), BorderLayout.WEST);
   add (new Button ("CENTER"), BorderLayout.CENTER);
}
```

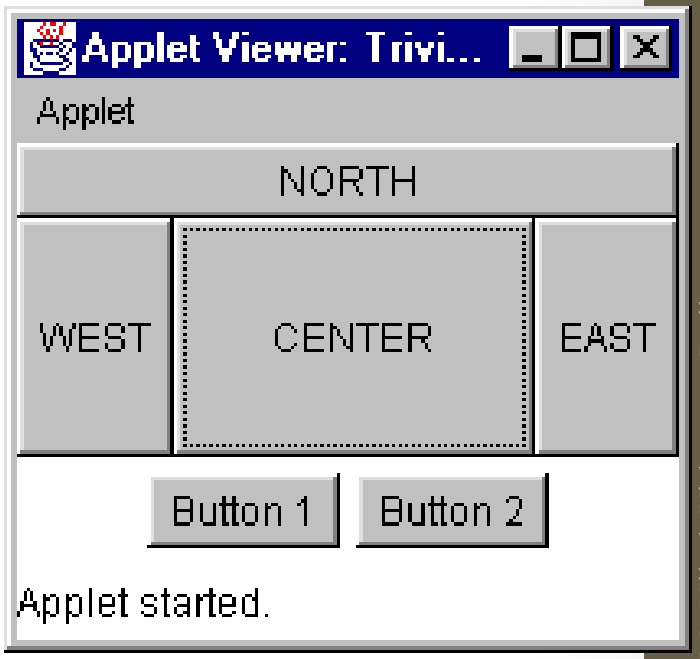
Complete example: BorderLayout

```
import java.awt.*;
import java.applet.*;

public class BorderLayoutExample extends Applet {
    public void init () {
        setLayout (new BorderLayout());
        add(new Button("One"), BorderLayout.NORTH);
        add(new Button("Two"), BorderLayout.WEST);
        add(new Button("Three"), BorderLayout.CENTER);
        add(new Button("Four"), BorderLayout.EAST);
        add(new Button("Five"), BorderLayout.SOUTH);
        add(new Button("Six"), BorderLayout.SOUTH);
    }
}
```

Using a Panel

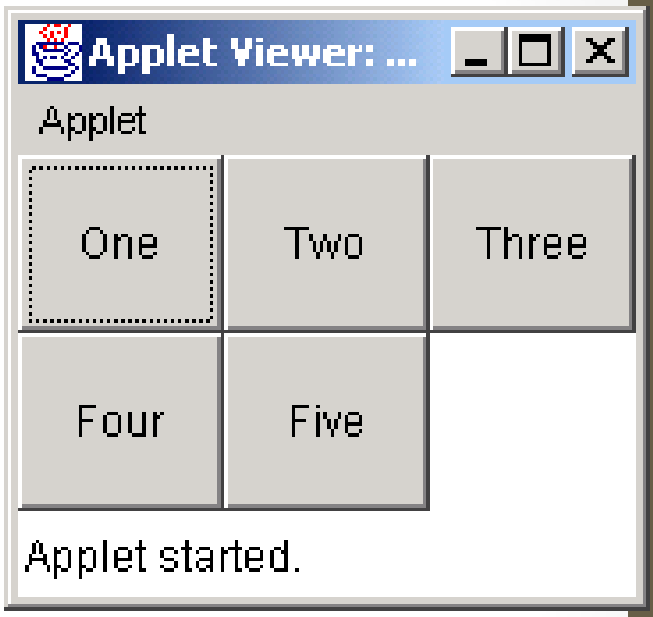
```
Panel p = new Panel();
add (p, BorderLayout.SOUTH);
p.add (new Button ("Button 1"));
p.add (new Button ("Button 2"));
```



GridLayout

- The GridLayout manager divides the container up into a given number of rows and columns:

  new GridLayout(*rows, columns*)

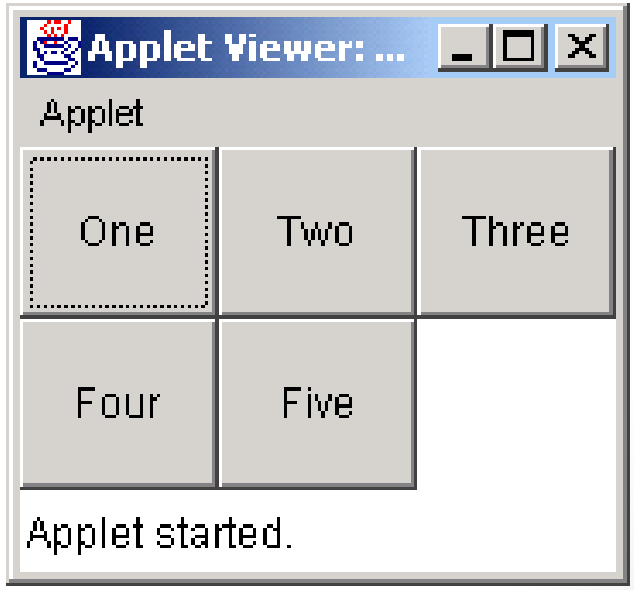


- All sections of the grid are equally sized and as large as possible

Complete example: GridLayout

```
import java.awt.*;
import java.applet.*;

public class GridLayoutExample extends Applet {
  public void init () {
    setLayout(new GridLayout(2, 3));
    add(new Button("One"));
    add(new Button("Two"));
    add(new Button("Three"));
    add(new Button("Four"));
    add(new Button("Five"));
  }
}
```



CardLayout

- The class **CardLayout** arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.
- Constructors:
  - **CardLayout()**
    Creates a new card layout with gaps of size zero.
  - **CardLayout(int hgap, int vgap)**
    Creates a new card layout with the specified horizontal and vertical gaps.

CardLayout

- Cards are typically held in an object of type **Panel**.
- Panel must have **CardLayout** selected as its layout manager.
- For Add component:
  - void add(Component *panelObj*, Object *name*);
- Methods:
  - void first(Container *deck*)
  - void last(Container *deck*)
  - void next(Container *deck*)
  - void previous(Container *deck*)
  - void show(Container *deck*, String *cardName*)

GridBagLayout

- **GridBagLayout** arranges components in a horizontal and vertical manner.
- GridBagLayout is the most complex and flexible of the standard layout managers.
- GridBagLayout, elements can have different sizes and can occupy multiple rows or columns.
- The position and behavior of each element is specified by an instance of the GridBagConstraints class.
- The actual grid size is based upon the number of components within the GridBagLayout and the GridBagConstraints of those objects.

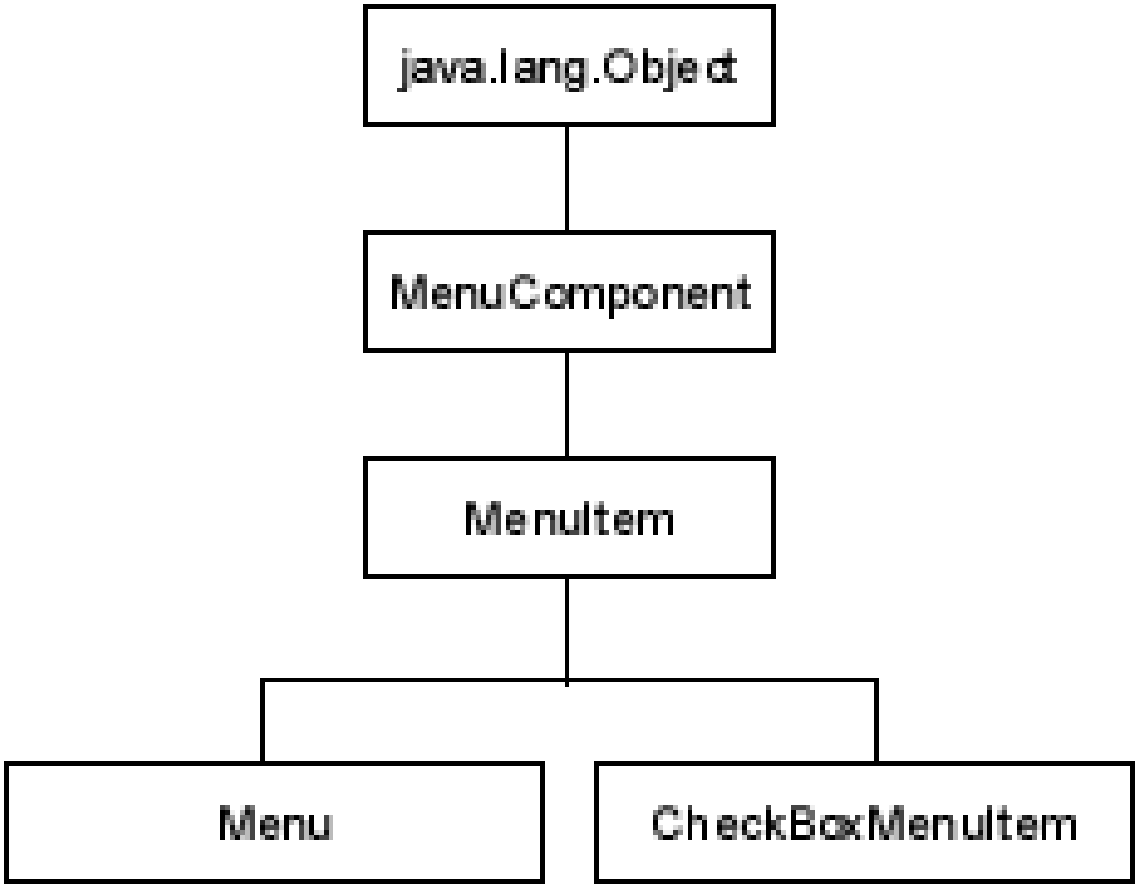
GridBagLayout

- Each GridBagLayout object maintains a dynamic rectangular grid of cells, with each component occupying one or more cells, called its *display area*.
- Each component managed by a grid bag layout is associated with an instance of GridBagConstraints that specifies how the component is laid out within its display area.
- Maximum capacity of a screen using GridBagLayout in
  - Java 1.0 is **128 128 cells**.
  - Java 1.1 is **512 512 cells.**
- *Constructor:*
  - **GridBagLayout()**

GridBagLayout

- For customize a GridBagConstraints object by setting one or more of its instance variables:
  - gridx, gridy:
    - Specifies the cell at the upper left of the component's display area, where the upper-left-most cell has address gridx = 0, gridy = 0.
  - gridwidth, gridheight:
    - Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1.
  - fill:
    - Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component.
  - a

MenuBar and Menu



MenuBar and Menu

- Top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices.
- Each choice is associated with a drop-down menu.
- Classes:
  - **MenuBar** : Contains one or more Menu objects
  - **Menu** : Contains one or more MenuItem objects
  - **MenuItem** : Object something selected by user.
- It is also possible to include checkable menu items
- These are menu options of type **CheckboxMenuItem** and will have a check mark next to them when they are selected.

MenuBar and Menu

- To create a menu bar, first create an instance of **MenuBar**.
- Set MenuBar using setMenuBar(MenuBarObject)
- Next, create instances of **Menu** that will define the selections displayed on the bar.
- Constructors:
  - Menu( )
  - Menu(String *optionName*)
  - Menu(String *optionName*, boolean *removable*)
- Individual menu items constructors:
  - MenuItem( )
  - MenuItem(String *itemName*)
  - MenuItem(String *itemName*, MenuShortcut *keyAccel*)

MenuBar and Menu

- Disable or enable a menu item by using:
  - void setEnabled(boolean *enabledFlag*)
  - boolean isEnabled( )
- Label set and get using:
  - void setLabel(String *newName*)
  - String getLabel( )
- Checkable menu item by using a subclass of **MenuItem** called **CheckboxMenuItem**. :
  - CheckboxMenuItem( )
  - CheckboxMenuItem(String *itemName*)
  - CheckboxMenuItem(String *itemName*, boolean *on*)

MenuBar and Menu

- Status about checkable MenuItem:
  - boolean getState( )
  - void setState(boolean *checked*)
- For add MenuItem:
  - MenuItem add(MenuItem *item*)
- For add MenuBar
  - Menu add(Menu *menu*)
- To get Item from Menu:
  - Object getItem( )

Dialog Box

- Dialog boxes are primarily used to obtain user input.
- They are similar to frame windows, except that dialog boxes are always child windows of a top-level window.
- Dialog boxes don't have menu bars.
- In other respects, dialog boxes function like frame windows.
- Dialog boxes may be modal or modeless.
- When a *modal* dialog box is active, all input is directed to it until it is closed.
- When a *modeless* dialog box is active, input focus can be directed to another window in your program.

Dialog Box

- *Constructors:.*
  - Dialog(Frame *parentWindow*, boolean *mode*)
  - Dialog(Frame *parentWindow*, String *title*, boolean *mode*)
- To create Dialog Box:
  - Create Frame or Applet
  - Create another class which extends Dialog class.
  - Call this new class from Frame/Applet class.
  - In constructor of Extended Dialog class, use super method and pass vales to constructor of Dialog.

FileDialog

- Java provides a built-in dialog box that lets the user specify a file.
- To create a file dialog box, instantiate an object of type **FileDialog**.
- Constructor:
  - FileDialog(Frame *parent*, String *boxName*)
  - FileDialog(Frame *parent*, String *boxName*, int *how*)
  - FileDialog(Frame *parent*)

Int how: **FileDialog.LOAD, FileDialog.SAVE**

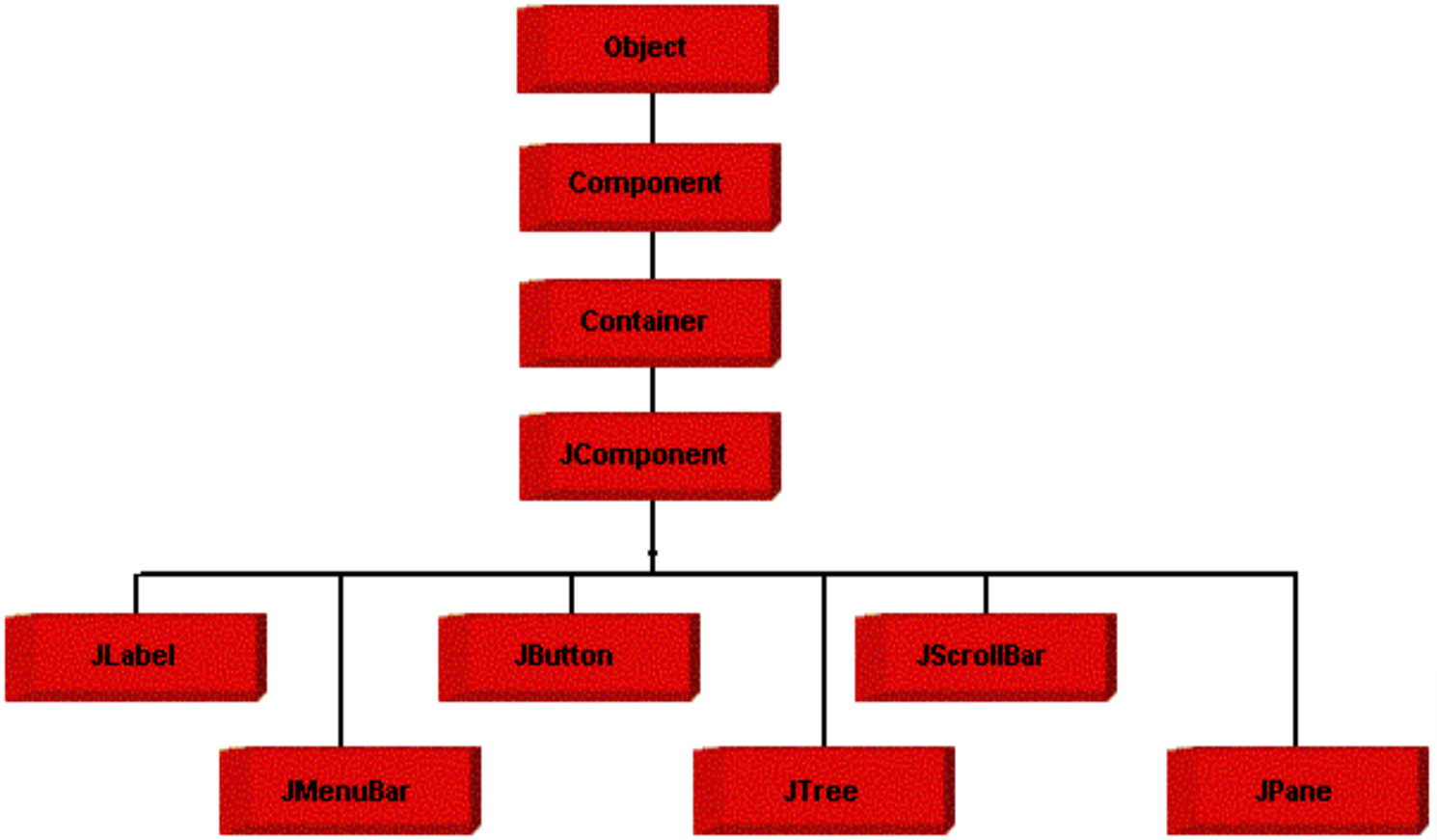
Methods:

String getDirectory( )
String getFile( )

Introduction to Swing

- Package : javax.swing.*
- Swing is set of classes which provides more powerful and flexible components as compare to AWT.
- Build on top of AWT API and a.cts as replacement of AWT API.
- Swing component follows a Model-View-Controller
- Swing Components are implemented using Java and so they are platform independent.
- Called lightweight components

Introduction to Swing



Introduction to Swing

- 100 % Java implementations of components.
- Use MVC architecture.
  - Model represents the data
  - View as a visual representation of the data
  - Controller takes input and translates it to changes in data

Difference Between AWT & Swing

- AWT uses Applet and Frame while Swing uses JApplet and JFrame for GUI.
- AWT is platform dependent code while Swing code is platform independent.
- Swing has bigger collection of classes and interfaces as compare to AWT.
- In Swing extra feature to Button: Provide Image.
- Swing provides: Tree, Table, Scrollpanes, Tabbedpanes etc new feature which not available in AWT.

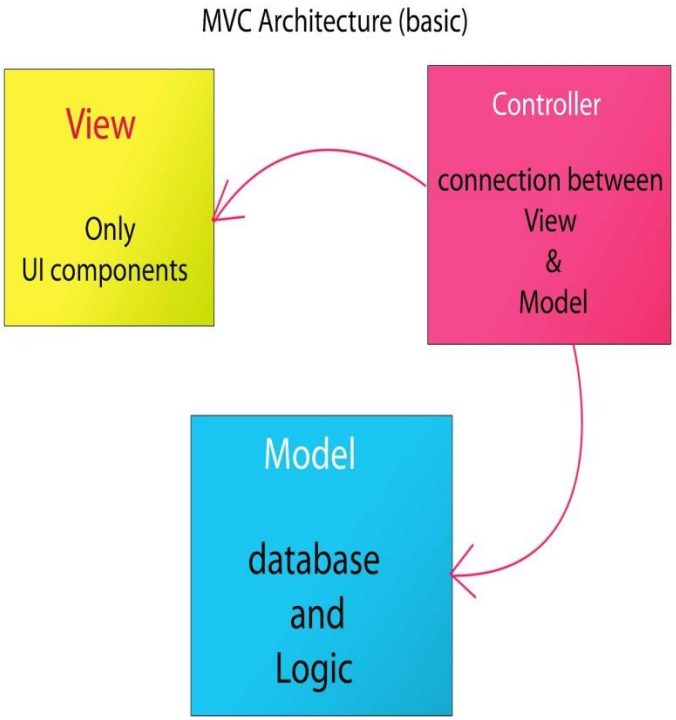
Important Classes by Swing

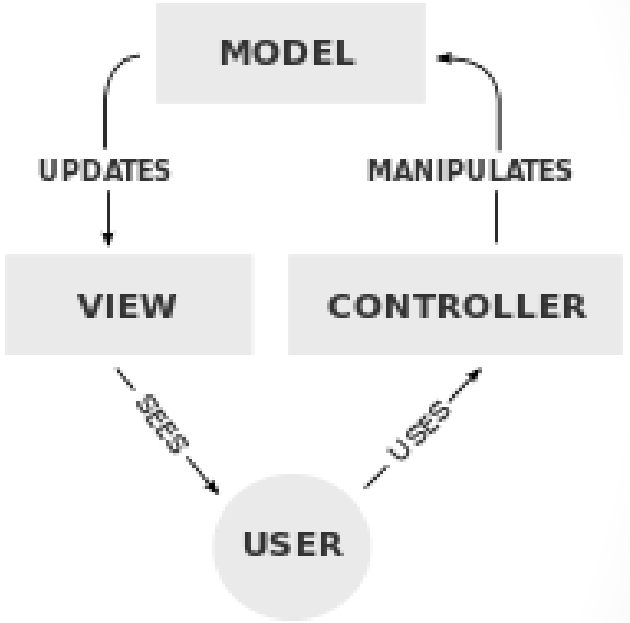
- Abstract Button
- ButtonGroup
- ImageIcon
- JApplet
- JButton
- JCheckBox
- JComboBox
- JLabel
- JRadioButton
- JScrollPane
- JTabbedPane
- JTable
- JTextField
- JTree

MVC Architecture

- Software design pattern for software development.
- Model:
  - Major function of this layer to maintain the data.
  - Database and logic.
- View:
  - Used to display full or partial data.
  - User Interface
- Controller:
  - Control the interaction and communication between Model and view.
  - Communication logic/integration logic

MVC Architecture





JApplet and JFrame

- Extends JApplet/JFrame class.
- Design UI in init() or Constructor method.
- Add all components on Container instead on JApplet/JFrame.
- getContentPane() method returns the container object.
- Call container add() method to add components.
- Guess default layout for JApplet/Jframe?
- For JFrame close operation: setDefaultCloseOperation()
- Parameters:
  - *DISPOSE_ON_CLOSE*
  - *EXIT_ON_CLOSE*
  - *DO_NOTHING_ON_CLOSE*

JLable and ImageIcon

- Small display area for text, image or both.
- Extends Jcomponent.
- Constructors:
  - JLabel(Icon *i*)
  - Label(String *s*)
  - JLabel(String *s*, Icon *i*, int *align*)
    align argument is either **LEFT**, **RIGHT**, **CENTER**,

- ImageIcon:
  - ImageIcon(String *filename*)
  - ImageIcon(URL *url*)

JLable and ImageIcon

- The **ImageIcon** class implements the **Icon** interface that declares the methods
  - int getIconHeight( )
  - int getIconWidth( )

- Other methods:
  - Icon getIcon( )
  - String getText( )
  - void setIcon(Icon *i*)
  - void setText(String *s*)

JTextField

java.lang.Object
    java.awt.Component
        java.awt.Container
            javax.swing.JComponent
                javax.swing.text.JTextComponent
                    javax.swing.JTextField

JTextField( )
JTextField(int *cols*)
JTextField(String *s*, int *cols*)
JTextField(String *s*)

AbstractButton

- Swing provide Icon with Button text.
- Swing buttons are subclasses of the **AbstractButton** class, which extends **Jcomponent.**
- **AbstractButton** contains many methods that allow you to control the behavior of buttons, check boxes, and radio buttons.
- Setter and Getter:
  - String getText( )
  - void setText(String *s*)

JButton

- Constructors:
  - JButton(Icon *i*)
  - JButton(String *s*)
  - JButton(String *s*, Icon *i*)

JCheckBox

java.lang.Object
- java.awt.Component
  - java.awt.Container
    - javax.swing.JComponent
      - javax.swing.AbstractButton
        - javax.swing.JToggleButton
          - javax.swing.JCheckBox

JCheckBox(Icon *i*)
JCheckBox(Icon *i*, boolean *state*)
JCheckBox(String *s*)
JCheckBox(String *s*, boolean *state*)
JCheckBox(String *s*, Icon *i*)
JCheckBox(String *s*, Icon *i*, boolean *state*)

JCheckBox

- void setSelected(boolean *state*)
- ItemEvent is generated.
- ItemListener interface is needed to handle ItemEvent.
- Public itemStateChnaged() used to override.

JRadioButton

java.lang.Object
- java.awt.Component
  - java.awt.Container
    - javax.swing.JComponent
      - javax.swing.AbstractButton
        - javax.swing.JToggleButton
          - javax.swing.JRadioButton

JRadioButton(Icon *i*)
JRadioButton(Icon *i*, boolean *state*)
JRadioButton(String *s*)
JRadioButton(String *s*, boolean *state*)
JRadioButton(String *s*, Icon *i*)
JRadioButton(String *s*, Icon *i*, boolean *state*)

JRadioButton

- ButtonGroup class is used to add radio button in group.
- ActionEvent is generated.
- ActionListener Listener interface is needed to handle ActionEvent.
- public void actionPerofrmed() used to override.

JComboBox

- Combination of text field and drop down list.
- Subclass of JComponet
- Only one entry can view at a time.
- Constructor:
  - JComboBox( )
  - JComboBox(Vector *v*)
- void addItem(Object obj): Used to add object in Combobox

JComboBox: EventHandling

- ItemEvent is generated.
- Implements ItemListener interface
- Override: itemStateChnaged(ItemEvent ie) method defined by ItemListener.

JTabbedPane

- *A tabbed pane* is a component that appears as a group of folders in a file cabinet.
- Each folder has a title.
- When a user selects a folder, its contents become visible.
- Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options.
- Subclass of JComponent

JTabbedPane

- *A tabbed pane* is a component that appears as a group of folders in a file cabinet.
- Each folder has a title.
- When a user selects a folder, its contents become visible.
- Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options.
- Subclass of JComponent

JTabbedPane

- Tabs are defined via the following method :
  - void addTab(String *str*, Component *comp*)
  - Str: title of pane
  - Comp: component, it can be JPanel
- Steps to create JTabbedPane:

1. Create a **JTabbedPane** object.
2. Call **addTab( )** to add a tab to the pane.
3. Repeat step 2 for each tab.
4. Add the tabbed pane to the content pane of the Japplet or JFrame

JScrollPane

- A *scroll pane* is a component that presents a rectangular area in which a component may be viewed.
- Horizontal and/or vertical scroll bars may be provided if necessary.
- Subclass of JComponent
- Constructor:
  - JScrollPane(Component *comp*)
  - JScrollPane(int *vsb*, int *hsb*)
  - JScrollPane(Component *comp*, int *vsb*, int *hsb*)
- Comp: Component, vsb and hsb: Scrollbar constant

JScrollPane

- HORIZONTAL_SCROLLBAR_ALWAYS
- HORIZONTAL_SCROLLBAR_AS_NEEDED
- VERTICAL_SCROLLBAR_ALWAYS
- VERTICAL_SCROLLBAR_AS_NEEDED

.

JTree

- A *tree* is a component that presents a hierarchical view of data.
- Trees are implemented in Swing by the **JTree** class, which extends **JComponent**.
- Constructors:
  - JTree(Hashtable *ht*)
  - JTree(Object *obj*[ ])
  - JTree(TreeNode *tn*)
  - JTree(Vector *v*)

JTree

- A **JTree** object generates events when a node is expanded or collapsed.
- The **addTreeExpansionListener( )** and **removeTreeExpansionListener( )** methods allow listeners to register and unregister for these notifications.
- Signature for these methods:
  - void addTreeExpansionListener(TreeExpansionListener *tel*)
  - void removeTreeExpansionListener(TreeExpansionListener *tel)*

JTree

- TreePath getPathForLocation(int *x*, int *y*):
- **TreePath** object that encapsulates information about the tree node that was selected by the user.
- The **TreeNode** interface declares methods that obtain information about a tree node.
- The **MutableTreeNode** interface extends **TreeNode**. It declares methods that can insert and remove child nodes or change the parent node.
- The **DefaultMutableTreeNode** class implements the **MutableTreeNode** interface. It represents a node in a tree.
- Constructor: DefaultMutableTreeNode(Object *obj*)

JTree

- To create a hierarchy of tree nodes, the **add( )** method of **DefaultMutableTreeNode** can be used.
  - void add(MutableTreeNode *child*)
- Tree Expansion event described by class:
  - **TreeExpansionEvent (Package: javax.swing.event)**
- The **getPath( )** method of this class returns a **TreePath.**
  - TreePath getPath( )
- **TreeExpansionListener** interface provides the following two methods
  - void treeCollapsed(TreeExpansionEvent *tee*)
  - void treeExpanded(TreeExpansionEvent *tee*)

JTree

- Steps to create Jtree:

1. Create a **JTree** object.
2. Create a **JScrollPane** object.
3. Add the tree to the scroll pane.
4. Add the scroll pane to the content pane of the applet.

JTable

- A *table* is a component that displays rows and columns of data.
- You can drag the cursor on column boundaries to resize columns.
- You can also drag a column to a new position.
- Subclass of JComponent
- Constructor:
  - JTable(Object *data*[ ][ ], Object *colHeads*[ ])
  - *data* is a two-dimensional array of the information
  - *colHeads* is a one-dimensional array with the column headings.

JTable

- Steps to create Jtable

1. Create a **JTable** object.
2. Create a **JScrollPane** object.
3. Add the table to the scroll pane.
4. Add the scroll pane to the content pane of the JApplet or JFrame.

Vocabulary

- AWT – The Abstract Window Toolkit provides basic graphics tools (tools for putting information on the screen)
- Swing – A much better set of graphics tools
- Container – a graphic element that can hold other graphic elements (and is itself a Component)
- Component – a graphic element (such as a Button or a TextArea) provided by a graphics toolkit
- listener – A piece of code that is activated when a particular kind of event occurs
- layout manager – An object whose job it is to arrange Components in a Container

The End